US-PAT-NO: 5918017

DOCUMENT-IDENTIFIER: US 5918017 A

TITLE: System and method for providing dynamically alterable computer clusters for message **routing**

---------- KWIC ---------

Abstract Text - ABTX (1):
A TCP-connection-**router** performs encapsulated clustering by dividing each encapsulated cluster into several Virtual EC (VECs), dynamically distributing incoming connections within a VEC based on current server load metrics according to a configurable policy. In one embodiment, the connection **router** supports dynamic configuration of the cluster, and enables transparent recovery which provides uninterrupted service to the VEC clients.

Application Filing Date - AD (1):
19960823

TITLE - TI (1):
System and method for providing dynamically alterable computer clusters for message **routing**

Brief Summary Text - BSTX (5):
An Encapsulated Cluster (EC) is characterized by a Connection-**Router** (CR) node and multiple server hosts providing a set of services (e.g. Web service, NFS, etc.). An example of a system which provides encapsulated clustering is described in U.S. Pat. No. 5,371,852, entitled "METHOD AND APPARATUS FOR MAKING A CLUSTER OF COMPUTERS APPEAR AS A SINGLE HOST ON A COMPUTER NETWORK".

Brief Summary Text - BSTX (12):
A further object of this invention is to provide means for a designated node to take over the operation of a failed connection-**router** in such a way that network clients experience no interruption of service.

Brief Summary Text - BSTX (15):
In a preferred embodiment, the EC can appear as (1) a VEC (a **single IP address** for all the remote clients) or (2) as multiple VECs (aliasing several IP addresses.) The TCP-CR node owns these IP addresses and receives all their TCP connection requests. Each IP address is associated with a VEC. The TCP-CR distributes new TCP connections to hosts within each VEC according to the weights associated with the VEC. The TCP-CR supports dynamic configuration

that allows: Dynamic definition of VECS. Dynamic configuration of the weights associated with a VEC. Automatic or manual management of VECS (adding or removing hosts, services, etc.). This solution allows for dynamic configuration, addition and removal of server hosts, while avoiding the problem of cached server names in the network.

Detailed Description Text - DETX (3):

This present virtual encapsulated cluster system can be embodied as an improvement to U.S. Pat. No. 5,371,852. U.S. Pat. No. 5,371,852, entitled "METHOD AND APPARATUS FOR MAKING A CLUSTER OF COMPUTERS APPEAR AS A SINGLE HOST
ON A NETWORK" (Ser. No. 960,742; filed Oct. 14, 1992; assigned to the same assignee as the present invention) is incorporated by reference herein as if printed in full below. FIG. 1 shows an embodiment of the encapsulated cluster invention of U.S. Pat. No. 5,371,852. Like the system of U.S. Pat. No. 5,371,852, the present system **routes** TCP information that crosses the boundary of a computer cluster. The information is in the form of **port** type messages. Incoming messages are **routed** and the servers respond so that each cluster appears as a single computer image to the external host. In the present system a cluster is divided into a number of virtual clusters (virtual encapsulated clusters). Each virtual encapsulated cluster appears as a single host to other hosts on the network which are outside the cluster. The messages are **routed** to members of each virtual encapsulated cluster in a way that keeps the load balanced among the set of cluster nodes.

Detailed Description Text - DETX (4):

FIG. 3 shows an embodiment of a Connection **Router** for the TCP family of protocols, the TCP-Connection-**Router** (TCP-CR) 300. The apparatus comprises two or more computer nodes (105-109) connected together by a communication link, called an interconnect 110, to form a cluster. (Note that in one embodiment of the invention, the interconnect can be a network.) One of the computers in the cluster, serving as a gateway 109, is connected to one or more external computers and/or clusters (hosts) through another communication link called a network 120. A gateway can be connected to more than one network and more than one node in the cluster can be a gateway. Each gateway connection to a network, i.e., boundary, can have multiple addresses on the network. Each gateway has a TCP-Connection-**Router** (TCP-CR) 300 which consists of a Manager 320 and an Executor 340 and an optional Recovery Manager as described in FIG. 10. The Manager controls the **routing** by sending command requests 344 to the Executor and evaluating the responses 346. The Executor consists of a message switch 140 similar to that of U.S. Pat. No. 5,371,852, and a VEC **router** 310.

Detailed Description Text - DETX (5):

FIG. 4 shows an alternate embodiment of the present invention. As in the preferred embodiment, the nodes 107 of the cluster communicate their responses directly back to the clients 130. However, in this embodiment there is no dedicated interconnect 110 (as shown in FIG. 3), all cluster nodes are connected by the external network 120. The TCP-Connection-**router** remains the same. A sample request 348 goes from a client 130 through the Gateway 109 and onto a cluster node 107 via the external network 120. The corresponding

response 350 goes directly from node 107, to the client 130 via the external network 120.

Detailed Description Text - DETX (7):

If the TCP-connection-**router** node 109 should cease to operate, all the nodes of the cluster will be unable to provide service to their remote clients. To address this problem we add a Recovery Manager which becomes active in the designated backup Gateway node when a functioning Gateway fails, and enhance the server nodes to keep recovery data. Clients need not take any action to recover from a Gateway failure, and continue to receive uninterrupted service from the cluster. A more detailed description of the Recovery Manager is presented in Section 4.

Detailed Description Text - DETX (9):

FIG. 5 shows the preferred embodiment of the executor 340. The executor consists of a command processor 540, message switch 140, and VEC **router** 310. The command processor 540 receives request for the executor 340 and returns responses 346. The command processor interacts with the message switch 140 and VEC **router** 310 to preform request and construct responses. The command processor may affect the connection table 510, VEC table 550, **port** table 520 or server table 530. The message switch 140 and connection table 510 are the same as the message switch and connection table of U.S. Pat. No. 5,371,852. In the preferred embodiment this invention the VEC **router** 310 does not modify incoming packets. Packets are forwarded to servers which have been configured so that responses will be sent directly to the clients from the internal nodes.

Detailed Description Text - DETX (10):

The message switch 140 is essentially the same as the message switch in U.S. Pat. No. 5,371,852. However, because of the present invention the message switch in the preferred embodiment has been optimized and an additional check has been added to the message switch. The message switch must check to see if the message is for a VEC known to the VEC **router**.

Detailed Description Text - DETX (11):

The VEC **router** keeps a set of addresses which represent each VEC to clients on the external network. The VEC **router** forwards requests to internal nodes of the cluster without modifying the received request. Each internal node of the cluster is associated wit one or more VES and only receives requests for VECs which it is associate with. Using techniques known to the art, in the present invention the internal nodes are configured to accept packets sent to the address representing a VEC and reply directly to the clients. In the prior art the message switch 140 had to rewrite packet headers for incoming request (FIG. 1 140) and rewrite packet headers for responses (FIG. 1 120) to request. In the present invention rewriting packet headers is not necessary. (The prior art can be used with the present invention.) The performance of the present invention is better than the prior art because packet headers are not rewritten and the response packets do not flow through the gateway node 109. Because response packets do not flow through the TCP-Connection-**router** the message switch does not receive any response packets from nodes internal to the

cluster. As a result in the preferred embodiment the header rewriting has been eliminated and checking for response packets from internal nodes has been eliminated.

Detailed Description Text - DETX (12):

A direct consequence of this improvement is that the VEC **router** only sees one half of the flows between the client and the internal node providing services. This makes it difficult to maintain an accurate connection table. To solve this problem the present invention uses two new timers specific to its connection table a stale timeout and a FIN timeout. Using these two timers and communication flows and timers known to the art, the connection table can be accurately maintained.

Detailed Description Text - DETX (13):

Connection table entries are considered to be in one of two states ACTIVE or FIN. Whenever a new connection is established a connection table entry is created and placed in the active state. Whenever a packet flows on a connection for which there is an entry in the connection table the connection entry is time stamped. When the VEC **router** seen a FIN flow from the client to the node providing services, the associated connection table entry is placed in the FIN state. (Packets may continue to flow on connections placed in FIN state.) A connection table entry is considered closed and available for purging when the amount of time identified by the FIN time out has expired since the last packet was forwarded from the client to the server on that connection. If the client fails without sending a FIN the connection record entry remains. The stale timeout specifies how long to wait after the last packet has flowed on an active conversation before purging the connection table entry.

Detailed Description Text - DETX (14):

FIGS. 7A-7C show the flow chart of the VEC **router** 310. In FIG. 7A the VEC **router** waits for a packet 702. When a packet is received the VEC **router** checks 704 whether the packet is for an existing TCP connection or is for a new TCP connection. If the packet is for an existing TCP connection then it checks 708 to see if the packet is a FIN, SYN, or RST (all packet types known to the art). If the packet is not one of these it forwards 722 the packet to the internal node associated with the connection. Otherwise, it checks 710 to see if the packet is an RST. If the packet is a RST, the conversation is purged from the connection table resetting the connection 712 and the packet is forwarded 722 to the internal node that was associated with the connection. If the packet is not an RST the VEC **router** 310 checks 714 to see if the packet is a SYN. If the packet is a SYN it established the connection 716 which brings the connection into active state even though the connection previously existed. The VEC **router** 310 then checks 718 to see if the packet is a FIN. If the packet is a FIN the connection is place in FIN state 720. After FIN processing or if the packet was not a FIN it is forwarded to the server associated with the connection 722.

Detailed Description Text - DETX (15):

FIG. 7B shows the non existing connection flow chart. When the check 704

finds a non existing connection. The VEC **router** first checks 724 to see if the packet is a SYN. If the packet is not a SYN it is discarded 726. If the packet is a SYN a connection is set 728 up in active state, a server is selected 730, and the packet is forwarded 722 to the server that was selected.

Detailed Description Text - DETX (16):

FIG. 7C shows the flow chart for the process of selecting a server 730 for a new connection. In the present invention this function implements the weighted **routing**. For the purposes of this discussion of selecting a server, the internal nodes of a VEC are considered to be numbered from one to n. For example if a VEC has seven nodes, the numbers are 1,2,3,4,5,6 and 7. For the purpose of this discussion of selecting a server eligible weights are considered to be numbered from the maximum legal value to one. For example if the max legal value is five the eligible weights would be 5, 4, 3, 2, and 1. Zero is a special value. Weights are also selected in decreasing order. The present invention associates a weight with each internal node providing a specific service. It guarantees for each service that at least one of the nodes has the maximum non zero weight or all of the nodes have zero weight.

Detailed Description Text - DETX (17):

The function which selects a server 730 first picks the number corresponding to the next highest server 734 and the current eligible weight. It then checks 735 to see if this number is too large. If the number is not to large it checks 746 to see if the server corresponding to this number is a good choice. (This check will be described more later.) If the number was too large it picks the first server 736 and the next lower weight. It then checks 738 to see if the next lower weight would be zero. If the next lower weight would not be zero it is used instead of the current eligible weight and this function checks 746 to see if the current server is a good choice. After selecting the first server and the maximum weight, this function checks 742 to see if there are any servers available to **rout** packets to. No servers are available when all of the available nodes have weight zero. If there are no available servers, the packet is returned 744 without selecting a server. If there are servers available this function checks to see if it has a good choice 746. A good choice is defined as a server whose weight is greater than or equal to the current eligible weight. If it is a good choice, the server is selected 748 and returned to the VEC **router** 750. If it is not a good choice then the algorithm picks the next server 734.

Detailed Description Text - DETX (19):

FIG. 8 shows an embodiment of the data structures used by the VEC **router**. The VEC table 550, contains the set of addresses which are the VEC addresses on the external network. All parameters which are associated specifically with a VEC are also contained in this table. Each VEC is associated with a **port** table 520 which contains the set of **ports** 802 that the VEC is providing service for. Each **port** entry 802 has associated with it a stale timeout 804, FIN timeout 806 and other **port** specific attributes 808. Each **port** has associated with is a subset of the internal nodes of the VEC which are used to provide the services associated with that **port**. The node table 530 contains addresses of the nodes 820 associate with the **port,** the current weight 822 associated with this node,

and other node specific information 830. (An example of node specific information is counters which indicate the number of connections in active state, the number of connection in FIN state, and the total number of completed connections.) The Node Table 530 also contains the state necessary for the function which selects a server to implementing weighted **routing** over the set of node in this table. The node table contains the total number of nodes 810, last chosen node 812, current eligible weight 814, maximum weight 816, and weight bound 818. The weight bound is used to limit the variance of the maximum weight. No node is allowed to have a weight greater than the weight bound.

Detailed Description Text - DETX (21):

The Connection-**Router**-Manager (Manager 320) invention is a method and apparatus for dynamically distributing incoming connections using several load metrics according to a configurable policy. The Manager provides a control loop that dynamically modifies the weights of the Executor 340 **routing** algorithm to optimize the allocation of cluster resources. The goal of this invention is to improve the overall throughput of the cluster and to reduce the aggregate delay of the service requests, by distributing incoming TCP connections according to the current state of the cluster. Hence, this invention describes a method to distribute the connections to the server hosts that improves the utilization of the servers and reduces the delay of serving the requests.

Detailed Description Text - DETX (22):

FIG. 6 shows a sample embodiment of the Manager 320 of the present invention within a cluster 600 of five nodes (105, 106, 107, 108, and 109). FIG. 6 uses the alternative network configuration of FIG. 4, but the configuration of FIG. 3 is also possible. One of the nodes is a gateway 109 which connects to an external network 120 and executes the TCP-Connection-**Router** 300 (the Executor 340 and the Manager 320). The Manager 320 consists of 5 generic components, a load manager (Mbuddy) 610, an external control interface (Callbuddy) 620, a cluster host metric manager (Hostmonitor) 630, a Forward Metric Generator (FMG) 640, and a User Programmable Metric Manager (UPMM) 650.

Detailed Description Text - DETX (23):

Mbuddy 610 can use four different classes of metrics to compute a weights function for the executor: input metrics, host metrics, service metrics, and user metrics. Mbuddy 610 receives these metrics and other relevant information from the Executor interface 346, the Callbuddy interface 624, the Hostmonitor interface 634, the FMG interface 644, and the UPMM interface 654. Mbuddy controls the weights associated with the executor **routing** algorithm for each VEC **port** server via interface 344.

Detailed Description Text - DETX (24):

Mbuddy 610 will periodically request from the Executor 340 the values of the internal counters associated with each server via interface 346. For example, it will periodically request the values of the counters of the total number of connections established for each server. By substracting two counters of a

server polled at times T1 and T2, Mbuddy 610 can compute a metric variable that represents the number of connections received during the time period T1-T2. The aggregation of such input metrics provide an approximation to the characteristic rate of connection requests for each VEC and each **port** service.

Detailed Description Text - DETX (26):

The Forward Metric Generator (FMG) 640 produces and evaluates application-specific or service-specific metrics using forward requests, that is, they originate at the Gateway 109 computer. The evaluation consists in producing appropriate requests for each of the cluster host servers and measuring their answering delays. For example, to obtain a forward delay metric on an HTTP server, the FMG may generate an HTTP "GET /" request to each HTTP server in the cluster serving a particular **port** (e.g. **port** 80). The FMG 640 will then measure the corresponding delays of servicing the HTTP request and forward a metrics vector to Mbuddy 610. If the request is not answered by a policy-specific threshold time, then the FMG will mark the corresponding service node as temporarily not receiving new requests of the particular service type. This information is used by the manager to decide that a service at a particular host is temporarily unreacheable, and hence no more connection requests of this type should be forwarded to it.

Detailed Description Text - DETX (29):

The Mbuddy 610 component is a load manager that establishes a dynamic feedback control loop between the servers and the Connection **Router** Gateway node. Mbuddy adjusts the weights of the Executor 610 **routing** algorithm so that servers which are lightly loaded according to the load metrics will receive a larger portion of the incoming TCP connections of their type. Given an arbitrary set of load and policy metrics as defined above, Mbuddy will compute a new relative weight for each server of each **port** in each VEC, based on its current metrics and its current weight.

Detailed Description Text - DETX (30):

The weight assignments are computed for each **port** on every VEC as follows: (1) Compute all the aggregate metrics (AM) for all the executing servers. (2) Compute all the current weight proportions for each executing server (CWP). (2) For each metric M compute for each server S the metric proportion (MP) of its value (relative to the aggregate AM). (3) For each server compute a new weight NW: (3a) If the server has been quiesced set is NW to 0. (3b) If the server has a sticky weight W use the value of W as the NW. (3c) compute a vector NWV, where each entry NWV[i] is based on a single metric M[i], by the following formula:

Detailed Description Text - DETX (39):

Messages (ip packets) arrive at the cluster gateway, directed to a particular TCP or UDP protocol **port**. The message switch within the Gateway allows a message **routing** function to be installed for a protocol **port**. The **routing** function is called for each message arriving for the associated **port,** and is responsible for selecting the internal node and **port** to which the message is forwarded. Information specifying the established connection and

the cluster node holding the connection is recorded in a table in the Gateway; this table is used by the message switch to **route** incoming packets on established connections to the correct cluster node.

Detailed Description Text - DETX (40):
Relatively static information, such as which server **ports** have installed message switch functions is maintained, and other manager configuration information is kept in a shared file, accessible to both primary and backup Gateway. Current connection information changes very rapidly and is managed according to the techniques described herein.

Detailed Description Text - DETX (41):
Each interior node 107 keeps a shadow 1010 of the Gateways **routing** table for its own connections (not for any connections to other nodes). This shadow table is used by the node to respond to the taking-over Gateway's request from the Recovery Manager 1020 in the backup Gateway 1030 during takeover. This table greatly reduces the amount of time the interior node needs to respond to the taking-over Gateway, and this is very important because, to keep established connections live, the takeover Gateway must be operational within the "time-out" period that the connection-based protocol allows for successful completion of a communication.

Detailed Description Text - DETX (42):
To reclaim space for entries in the connection table, both in the Gateway and in the shadow kept at the interior node, we proceed as follows. Connections are either in one of two states, Active or FIN. Connection table entries are time stamped on every reference. A user configurable timer called the FIN.sub.-- TIME.sub.-- OUT is kept. This timer represents the point in time after the last reference to a conversation in FIN state that it will be assumed to be closed. The timers can be either global, per service address, or per **port**. The intent of active close (one side of the connection has sent FIN but the other continues to send on the connection) is that the server would be allowed to continue sending data to a client and at the end of the data transmission the conversation would be closed. The client is allowed to actively close the conversation as a means of telling the server that no further request from the client will be sent. For the purposes of this discussion we will assume that the client's request are being sent through the **router**. This protocol works because the server continues to send data which will be acked. The **router** and consequently the server will see the acks and continually time stamp the connection table entries. Once the server completes sending data to the client and closes its "half" of the conversation the final ack will flow to the server from the client. After FIN.sub.-- TIME.sub.-- OUT time has elapsed the server can purge the connection entry. A second timer, STALE.sub.-- TIME.sub.-- OUT is kept by the Gateway. Any connection which is in Active state with no activity for longer than STALE.sub.-- TIME.sub.-- OUT can be purged.

Detailed Description Text - DETX (47):
(2) the backup Gateway interrogates each functioning node of the cluster,

requesting descriptions of all UDP **ports** allocated at the respective node, and TCP connections established through the primary Gateway between itself and hosts outside the cluster; the backup Gateway does this using a private ip-based protocol. The shadow connection table kept in each node allows immediate response from the nodes, increasing the probability that established connections do not time out during Gateway takeover. The algorithm described above for recognizing closed connections and reclaiming the space used to support them minimizes the size of the shadow connection table and contributes to reducing the time required to accomplish Gateway takeover.

Detailed Description Text - DETX (48):
(3) the backup Gateway records responses from each functioning node, and records the node's UDP **ports** and TCP connections in the connection table 510 of FIG. 5 in the backup Gateway's Executor 340 FIG. 10.

Claims Text - CLTX (1):
1. A method for providing for **routing** of a plurality of incoming messages, each message having a message header, across a boundary of a cluster of computer nodes, the cluster coupled to one or more networks and having tables of subsets of said nodes, comprising the steps of:

Claims Text - CLTX (2):
routing a plurality of messages, wherein said **routing** comprises, for each message of said plurality of messages:

Claims Text - CLTX (3):
locating and reading a **port** number and target address in the message header of a **port** type message;

Claims Text - CLTX (5):
based on the **port** number, selecting a function which determines a **routing** destination for a message from a plurality of possible destinations in the subset, the **routing** destination being a computer node in the subset;

Claims Text - CLTX (6):
sending the message to the **routing** destination; and

Claims Text - CLTX (7):
while the plurality of messages are being **routed** across the boundary, dynamically altering at least one of membership in a subset and the number of subsets by manipulating entries in said at least one table.

Claims Text - CLTX (8):
2. The method of claim 1 wherein the selecting is based on the **port** number and a protocol identifier in the message header.

Claims Text - CLTX (12):
6. A gateway for transparently **routing** incoming messages, each having a message header, across a cluster of computer nodes, each of said nodes having an identifier, wherein said cluster is coupled to one or more networks, comprising:

Claims Text - CLTX (13):
means for locating and reading a **port** number and target address in the message header of a **port** type message and based on the target address, for selecting a subset of the computer nodes;

Claims Text - CLTX (14):
means for selecting a function based on the **port** number, wherein the function determines a **routing** destination for the message from a plurality of possible destinations in the subset, the **routing** destination being a computer node in the subset;

Claims Text - CLTX (16):
automatic means for dynamically altering at least one of membership in a subset and the number of subsets by manipulating entries in said tables during message **routing**.

Claims Text - CLTX (20):
10. A method for **routing** incoming messages each having a message header across a boundary node of a cluster of computer nodes, the cluster connected to one or more networks, comprising the steps of:

Claims Text - CLTX (22):
locating and reading a **port** number in the message header of a **port** type message;

Claims Text - CLTX (23):
based on the **port** numbers selecting a function which determines a **routing** destination for the message from a plurality of possible destinations, the **routing** destination being a computer node in the cluster;

Claims Text - CLTX (30):
locating and reading a **port** number and target address in the message header of a **port** type message;

Claims Text - CLTX (32):
wherein the **routing** destination is selected from only the subset.